Insight:

Method lookup

When you call a method on an object, Ruby follows predefined set of rules to find exact method to ran. Knowing this rules will help a lot in your metaprogramming journey.

Method lookup

Methods are looked up in the object itself.

1) In the object

```ruby
class MyClass
    def foo
        "Method foo in MyClass"
    end
end

obj = MyClass.new

def obj.foo
    "Singleton method foo"
end

obj.foo                                  # "Singleton method foo"
```

Method lookup

Methods are looked up in the object itself.

If not found:

- Methods are looked up in the object's class.

2) In the object's class

```
class MyClass
      def foo
            "foo"
      end

      def self.bar
            "bar"
      end
end

obj = MyClass.new
obj.foo
obj.bar

MyClass.foo
MyClass.bar
```

Method lookup

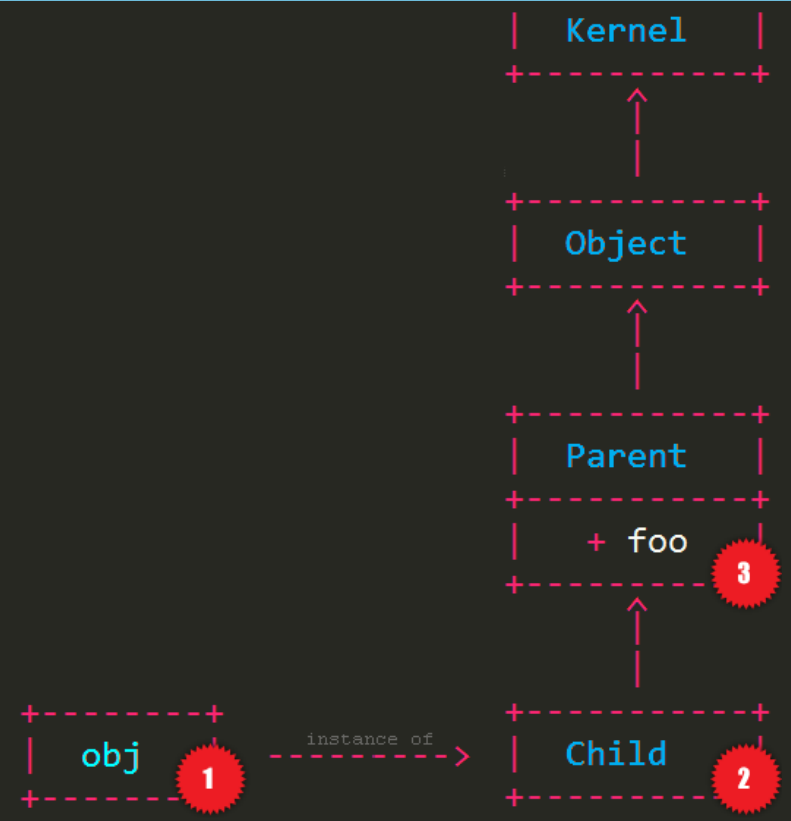
Methods are looked up in the object itself.

If not found:

1. Methods are looked up in the object class.
2. Methods are looked up in class's ancestry tree.

3) In the object's class ancestry chain

```ruby
class Parent
    def foo
        "foo in parent"
    end
end

class Child < Parent
end


obj = Child.new
obj.foo                          # "foo in parent
```



Method lookup

Methods are looked up in the object itself.

If not found:

1. Methods are looked up in the object class.
2. Methods are looked up in class's ancestry tree.
3. Ruby calls `method_missing()` method on the object.

3) method_missing() is called on the object

```ruby
class MyClass
    def method_missing(method_name, *args, &block)
        if method_name == :foo
            # Method implementation
            "Hurray, I’m not missing!"
        else
            super
        end
    end
end

obj = MyClass.new
obj.foo                                  # "Hurray, I’m not missing!"
obj.bar                                  # NoMethodError: undefined method `bar' fo
```

3) method_missing() is called on the object (cont.)

```ruby
class Parent
    def method_missing(method_name, *args, &block)
        if method_name == :foo
            "method_missing in parent"
        else
            super
        end
    end
end

class Child < Parent
end


obj = Child.new
obj.foo                                  # "method_missing in parent"
```

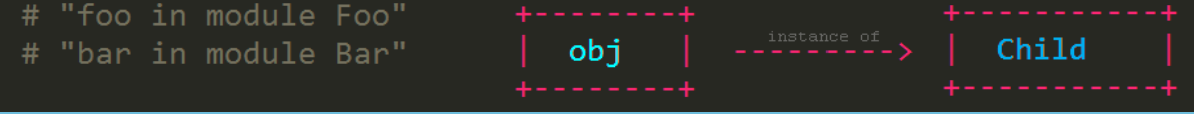
Object#method_missing()

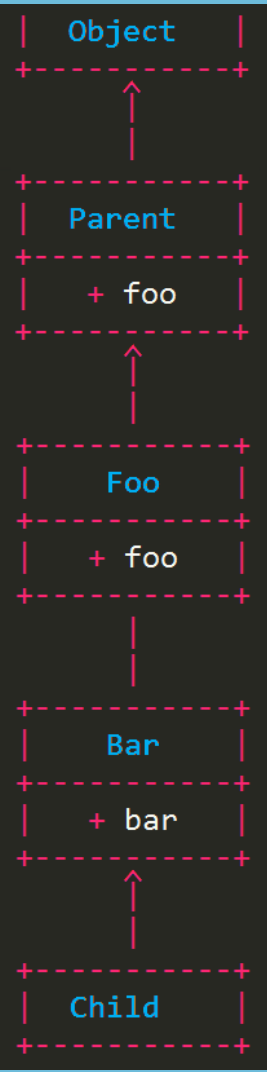
```ruby
class Object

    def method_missing(method_name, *args, &block)
        raise NoMethodError, "undefined method `#{method_name}' for #{self.inspect}"
    end

end
```

Method lookup and included modules

```ruby
class Parent
    def foo
        "foo in Parent"
    end
end

module Foo
    def foo
        "foo in module Foo"
    end
end

module Bar
    def bar
        "bar in module Bar"
    end
end

class Child < Parent
    include Foo
    include Bar
end

obj = Child.new
obj.foo
obj.bar
```

```ruby
# "foo in module Foo"
# "bar in module Bar"
```





Method lookup

Methods are looked up in the object itself.

If not found:

1. Methods are looked up in the object class.
2. Methods are looked up in class's ancestry tree.
3. Ruby calls `method_missing()` method on the object.